Confidential High-Performance Computing in the Public Cloud

Keke Chen
Computer Science Department, Marquette University

***Abstract*—High-Performance Computing (HPC) in the public cloud democratizes the supercomputing power that most users cannot afford to purchase and maintain. Researchers have studied its viability, performance, and usability. However, HPC in the cloud has a unique feature – users have to export data and computation to somewhat untrusted cloud platforms. Users will either fully trust cloud providers to protect from all kinds of attacks or keep sensitive assets in-house instead. With the recent deployment of the Trusted Execution Environment (TEE) in the cloud, confidential computing for HPC in the cloud is becoming practical for addressing users' privacy concerns. This paper discusses the threat models, unique challenges, possible solutions, and significant gaps, focusing on TEE-based confidential HPC computing. We hope this discussion will improve the understanding of this new topic for HPC in the cloud and promote new research directions.**

CONFIDENTIAL COMPUTING preserves the confidentiality of data and computation while running programs on an untrusted platform, such as a public cloud. With the growing availability of high-performance computing (HPC) in the public cloud, we foresee that confidential computing will also be a need for potential HPC users who cannot access traditional HPC facilities. However, the study on the challenges and solutions for this topic is seriously lagging.

Traditionally, HPC facilities are maintained by national labs or major research institutes and accessed by authorized users. While many industrial users [1] and small-institute users are potential HPC users, they may find it cumbersome to access such exclusive resources due to policies and restrictions. Since purchasing and maintaining an HPC cluster is expensive, HPC in the public cloud is probably the most viable option for such cash-strapped users. To meet this unique demand, most major cloud providers have started offering HPC services. Researchers have done extensive studies to understand the problems with HPC in the public cloud, e.g., on viability, performance, and usability [1]. However, no sufficient studies have been done on confidentiality issues.

In non-cloud HPC environments, the integrity of data and computing has been the primary concern in HPC security, and the HPC provider is fully trusted to guarantee the security of data and computation. Studies have been done on issues such as hardware root of trust, software and data supply chain security, and identity management. However, confidential processing of sensitive assets, including data and possibly algorithms, is a unique feature and will be an emerging demand for outsourced HPC applications. Specific examples may include but are not limited to intellectual property protection, data or algorithm embargo, and legal requirements

[1] https://www.top500.org/news/why-we-care-about-industrial-hpc/

 Published by the IEEE Computer Society

on private data. Due to the concerns about curious or malicious insiders, co-tenants, and external attackers [2], users have hesitated to move sensitive data and computation to the public cloud.

Confidential computing techniques are becoming more practical in recent years due to the Trusted Execution Environment (TEE) development. TEE creates a *secure enclave* for running programs securely with the specific CPU instructions [3]. Most recent Intel, AMD, and ARM CPUs have implemented the TEE concept. Many cloud providers have started to provide TEE-enabled servers, e.g., Azure has Intel SGX-enabled servers, and Google provides AMD SEV servers. TEE essentially moves the trust on cloud service providers to the CPU manufacturers and reduces the attack surface from the entire software stack to the enclave. The hardware-enabled features have significantly improved the performance over pure software-based cryptographic approaches [4]. Typical TEE applications (without handling side-channel attacks) cost only about 1.x of non-TEE applications' [5]. During the past few years, confidential computing has been rapidly transformed from academic research to practical applications (e.g., fortanix.com), enabling new forms of computing and sharing with reduced risk of data breaches[2]. However, the combination of TEE-based confidential computing and HPC in the cloud remains an insufficiently explored area.

This paper will discuss the threat models, potential challenges, and solutions for applying TEEs to public HPC clouds. Other studies may have covered interesting topics around "HPC in the cloud", e.g., applying cloud computing technologies to manage a traditional HPC[3]. Our study is distinct from those, as we will focus on the fundamental issues in public HPC clouds – users' confidentiality and ownership concerns about their data and computation.

The remaining sections are organized as follows. Section "Thread Modeling" discusses threat models for confidential HPC in the public cloud. We review existing confidential computing solutions in Section "Types of Confidential Computing Solutions". Section "TEE for HPC in the Public Cloud" focuses on applying TEEs to public HPC clouds, including the challenges and solutions. Finally, we conclude the discussion.

Threat Modeling

To discuss the possible challenges and solutions, we will need to establish a clear context for applying confidential computing for HPC in the public cloud. We will focus on unique issues that distinguish public HPC cloud from traditional HPC.

Single-User Case. Users may run confidential computation tasks in an untrusted cloud server, where the server's OS or hypervisor can be compromised. The goal is to preserve data and program integrity and confidentiality while availability is out of concern. A typical TEE, such as Intel SGX, provides a hardware-protected memory area, i.e., the *enclave* [3], and guarantees the integrity of the data and computation running inside the enclave. While adversaries cannot directly access the enclave, they can still glean information via side channels, such as memory access patterns and CPU caches. However, cache-based attacks target all CPUs (regardless of having TEEs or not) and thus need manufacturers' micro-architecture level fixes. In contrast, the exposure of memory access patterns is inevitable as enclaves have to interact with the untrusted memory area. It's also reasonable to assume that attackers cannot access the cloud server physically, e.g., attaching a device to the server or access the motherboard, which exclude all attacks based on physical accesses. Figure 1 illustrates the threat model.

Collaborative-Multiparty Case. HPC applications often involve collaborative workflows, where the use of TEEs may enable new types of attacks. The following discussion also addresses general concerns with collaborative workflows, not specific to HPCs. We model a collaborative workflow as a directed graph consisting of the modules (data

[2]https://docs.microsoft.com/en-us/azure/confidential-computing/use-cases-scenarios

[3]https://www.hpcwire.com/2018/02/15/fluid-hpc-extreme-scale-computing-respond-meltdown-spectre/

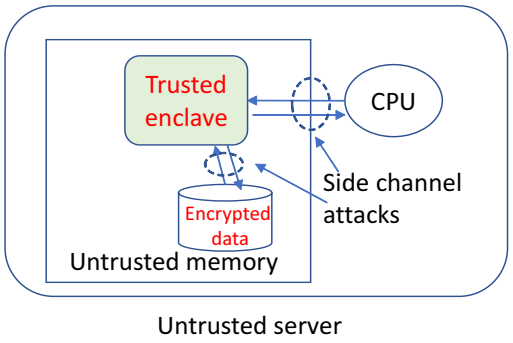

Figure 1. Threat model for TEEs.

sources or processing modules) contributed and shared by different participants, some of which are *confidential components*. Figure 2 shows two cases of confidential components in a collaborative workflow, where a participant P_1 holds a private dataset D_3 and a private algorithm C_1, and another participant P_2 has a private algorithm C_2 only, while all other components are public. More generally, we identify the following critical scenarios: (1) private datasets as the input or output of a processing component (either confidential or non-confidential), and (2) confidential processing components.

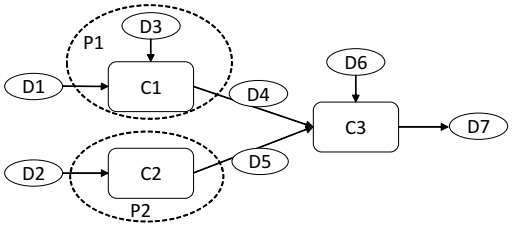

Figure 2. Collaborative workflow with private components. P_i: participants who may own private components, C_i: processing component, D_i: data component. P_1 and P_2 own private components, while all other components are public.

Reproducibility is critical for scientific workflows. A *reproducible workflow* must have a logging component to keep track of workflow provenance. Most workflow management systems, such as Galaxy and Taverna, can automatically log activities behind the scene. In contrast, users of manually built scripts and pipelines depend on public repositories, such as GitHub, to support reproducibility, using workflow scripts + a Readme file describing the inputs/outputs of each step. We consider *logging*, *provenance data analysis* (for debugging and optimization, etc.), and *reproducibility verification* are the minimal core *service components* in a reproducible workflow. These service components are likely moved to the cloud for better scalability, which can also benefit from confidential computing.

Based on the reproducible workflow model, we aim to protect two types of assets. (1) the confidentiality and integrity of private components, and (2) the integrity of service components. Intrinsically, protecting one type of asset may interfere with the other. We consider the following potential adversaries in the collaborative environment.

- Curious participants in the workflow. While the participants' major goal is to collaboratively generate results, they might be interested in learning the private data or algorithms.
- Dishonest owners of private components. They are also participants with demands on confidential processing. However, they may also take advantage of the confidential computing mechanism to disguise their fraudulent activities.

We can assume all the service components are running in a trusted environment, e.g., TEE enclaves, to make the attacking surface smaller. However, the *interplay* between the private components and service components still creates new challenges.

Types of Confidential Computing Solutions

We briefly review the available solutions of confidential computing and check whether they fit HPC applications.

Pure Software Approaches. For many years, researchers have studied the software approaches to achieve confidential computing [4]. We summarize them with the following categories.

- *Homomorphic Encryption* allows computations with encrypted data without decryp-

tion, which is ideal for computing on untrusted platforms such as public clouds. Fully homomorphic encryption (FHE) [6] allows any function to be implemented on encrypted data. However, FHE's high costs in implementing multiple levels of multiplication are the primary issue, despite recent improvements in ring-based implementations [6]. Additive homomorphic encryption (AHE) and somewhat homomorphic encryption (SHE) methods are more efficient than FHE schemes, while allowing only a small number of homomorphic multiplications.

- *Secure Multiparty Computation (MPC)* is another approach for multiple parties collaboratively working to evaluate a known function of their inputs while keeping the data private. Garbled Circuits (GC) and secret sharing among the main MPC methods. Recent advances such as FastGC [7] have also significantly reduced the cost of GC. However, they are still costly, as shown in several applications [8], [9]. Optimized secret sharing has been applied in confidential machine learning [9], which also suffer from high communication costs.
- *Hybrid Constructions* combines AHE, SHE, and multiparty computation primitives to minimize the overall costs of protocols. A few recent studies [9], [8] have shown such hybrid approaches are possible for data analytics, although the costs are still much higher than plaintext approaches and TEE based solutions.

Trusted Execution Environment. TEEs depend on unique CPU features to allow user-specified code and data to run inside a secure enclave that even a compromised OS or hypervisor cannot breach. It is an ideal hardware-level primitive for securely running programs on top of untrusted platforms, such as public clouds, edges, and third-party service providers. The most well-known TEE is Intel Software Guard Extensions (SGX), available in most Intel server processors, starting from the Skylake CPUs in 2015. AMD EPYC CPUs (since 2017) have also included the Secure Encrypted Virtualization (SEV) feature, which makes each protected virtual machine a secure enclave. Typically, a TEE automatically encrypts memory pages when they are not used (e.g., when swapped to the disk). The encrypted memory pages are decrypted and put in a protected memory area (e.g., the enclave page cache (EPC)) that only the owner process can access. AES encryption is used to make sure good performance and strong protection.

Let's take a closer look at the most popular TEE implementation: Intel SGX. SGX implementation reserves a region of the existing system memory called Private Reserved Memory (PRM). Intel extended their x86 instruction set to isolate PRM accesses from operating systems, virtual machines, or other privilege system codes. When the user wants to perform a secure computation, it creates an isolated container known as *enclave* and executes the confidential code inside the enclave. An enclave uses PRM to host data and code. Before creating an enclave, an Intel service can challenge the cloud provider via a three-party remote attestation protocol that verifies if the provider is using a certified SGX supported CPU. After creating the enclave, the user can safely upload their code to the enclave. Then, the user can pass encrypted data into the enclave, decrypt it, compute with plain text data, encrypt the result, and return it to the untrusted cloud components. During the runtime of an enclave, when other applications want to access the enclave memory, the CPU will deny such operation and return 0xFF, also known as abort page in SGX. An SGX application typically contains the untrusted part and the enclave part. Figure 3 shows SGX runtime interactions between these parts. Readers can check [3] to understand the detail.

Discussion. Enhanced by the hardware support, TEE programs can achieve much better performance than the pure software cryptographic approaches. The performance gain comes from three aspects. First, AES does not increase the ciphertext size and is much faster than homomorphic encryption methods in decryption and encryption

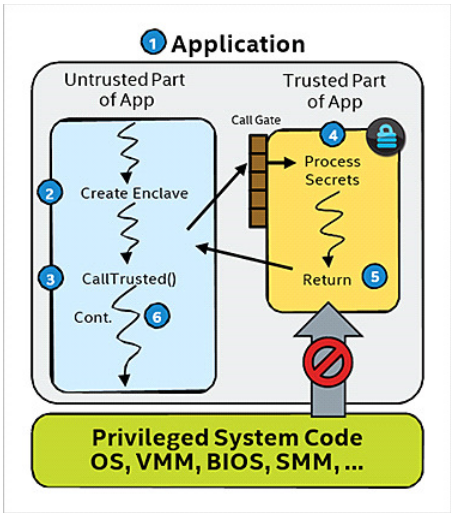

Figure 3. Illustration of SGX runtime execution (from intel.com)

operations. 128-bit AES is typically used in TEEs. In contrast, homomorphic encryption keys are typically 1024 or 2048 bits, resulting in long ciphertext and more expensive operations. Second, TEEs have much lower communication costs than MPC solutions. TEEs only require initial remote attestation to verify the authenticity of enclaves and programs. However, MPC incurs communication costs between two cloud servers for each basic computation step (e.g., addition and multiplication operations). Finally, the computation within the enclave is done with plaintext. It's thus much faster than computation with HE or MPC.

All these methods can preserve data confidentiality well. However, software cryptographic approaches do not protect code confidentiality, while we can also implement code confidentiality with TEEs. Both types of methods still suffer from side-channel attacks, which will be discussed in more detail later for TEEs. In contrast, protecting side channels is a lower priority in the research of pure software approaches, as other issues, such as performance and protocol-level security, have not been fully addressed yet.

TEE for HPC in the Public Cloud

As pure software cryptographic approaches take much more costs than the TEE approach, we believe the TEE approach is more practical for HPC applications in the cloud. There are still several challenges to using TEEs for typical HPC users. We summarize the challenges and discuss possible solutions in this section.

Unique Challenges

While TEEs guarantee good performance and strong security, several unique challenges exist. We summarize the main ones: usability, performance, side-channel attacks, and attacks in collaborative workflows.

Usability. Developing TEE applications may not be straightforward. For example, the code needs to be redesigned for Intel SGX: the application has to be split into two parts: the enclave program and the program in untrusted memory. Also, the enclave part of the code cannot use OS API directly to ensure a strong security guarantee. The learning curve will be steep for normal HPC users unfamiliar with the security concepts and the particular programming paradigm. AMD SEV does not require applications to be redesigned. However, if a higher level of confidentiality is desired, i.e., making programs resilient to side-channel attacks, the developer must modify the applications. Revising existing code is particularly unfriendly to HPC applications as most depend on low-level scientific computing libraries that have been used for decades.

TEE Side-Channel Attacks. Since most kinds of possible attacks in the conventional environment are no longer possible in TEEs, researchers focus on side-channel attacks. Memory side channels, i.e., access patterns, are the major ones for data-intensive processing. To be processed, encrypted data must be loaded from the untrusted areas, such as the non-TEE memory area or the file system, and then fetched by the enclave programs inside the TEE. Thus, interactions between the TEE and the untrusted area always exist regardless of what type of TEE is used, and they will be observed by adversaries and utilized to infer sensitive information. Oblivious RAM (ORAM) [10] has been a popular method to hide block-level access patterns for TEEs, such as ZeroTrace [10], Obliviate,

and Oblix. Researchers have also used page-fault interrupts, and page table features to extract secrets such as encryption keys inside enclave [11]. The most popular method to address this problem uses the CMOV instruction to rewrite each branching statement [12], [10], [13] to make them oblivious. The CPU cache is another popular side channel. Attacks like Meltdown and Spectra [14] apply to both Intel and AMD CPUs, and TEEs are not immune to such attacks. However, the defenses against cache-related attacks often depend on manufacturers' micro-architecture level firmware or software fixes.

Performance. In general, TEE will have a performance penalty depending on types of workloads and CPUs. Akram et al. [5] have shown that with proper configurations the cost for HPC benchmarks can be around x1.15 slowdown for AMD SEV, while varying in a larger range for Intel SGX. Note that these tests do not consider any side-channel protection mechanism. The current access-pattern protection mechanisms, such as ORAM will significantly increase the overhead – reducing the cost of protection has been one of the primary goals for ongoing research [13].

Issues with Collaborative Workflow. As HPC applications typically involve workflows, some of which may also include multiple parties, confidential computing in this context also raises new problems.

- *Owner's Attacks.* Private components may create a blind spot in the workflow system: They do not allow other users to examine the internal details, and the logging service only records the external information about using the algorithm component, i.e., parameter settings, input, and output. Thus, dishonest private-component owners have a chance to issue a *replacement attack* that the owner can replace the private algorithm or data anytime. This attack is even more challenging to detect for algorithms with randomization steps, which are common in scientific computing. A dishonest owner can forge a fake algorithm that works only for specific input-output pairs while replacing it later with another one that generates outputs with a similar statistical property.
- *Conflict between Confidentiality and Provenance Analysis.* Provenance analysis needs to access the log data, which includes the description of the activities around a private component (a dataset or a processing program). As a result, attackers may utilize this information to infer the content of the private component. For example, model-inversion attacks [15] are possibly applied to infer private input data from a known processing algorithm and output data; and model-stealing attacks [16] try to rebuild the private processing algorithm based on sample input-output pairs.
- *Reproducibility Verification* is a replay of workflow execution, supposedly conducted by an authorized third party. Due to the security requirements (e.g., passing secret keys to the enclave), TEE-based private components are controlled and executed by their owners, which have effectively prevented any unauthorized verification. However, it's inconvenient to demand all owners staying online for verification. Another concern is that the dishonest owner's attack can also be applied in this stage.

Possible Solutions

The application of the TEE approach is still at the early stage. In the following, we discuss some solutions that can be applied to HPC applications in the cloud.

Improving Usability. A few efforts have been conducted to usability issues of SGX. To avoid modifying existing applications, Graphene-SGX [17] and SCONE [18] try to build a library OS or a shim layer to allow unmodified Linux applications running inside enclaves. However, this approach does not protect access patterns from attacks, and it's difficult to incorporate any application-level protection methods into these frameworks. Other approaches, such as Google Asylo and Open Enclave, try to simplify SGX programming with an easier programming framework or library, so that users do not need to learn

the complex native SGX APIs.

For distributed data-intensive processing, VC3, M2R, and Opaque [19] try to adapt existing popular data-processing software stacks such as Hadoop and Spark by slightly modifying the original software, e.g., only moving the confidential data processing part into the enclave. However, leaving the system components in the untrusted memory area enables many attacks.

Gaps: To our best knowledge, all these usability-oriented projects do not address the side-channel attacks. However, these methods are particularly useful for legacy HPC applications, if side-channel protection is not a concern.

Protecting Block Access Patterns with ORAM. The ORAM-based approaches [10] try to disguise block I/O accesses and provide a generic block I/O interface for applications. However, ORAM-based methods have several notable drawbacks. (1) They are expensive. Each block access incurs $O(\log n)$ additional block accesses to disguise the actual access, where n is the number of blocks in the file. (2) Not all block access patterns leak sensitive information, which requires users to examine the application-specific block access patterns to design solutions with good performance, which is time-consuming and error-prone. (3) As a low-level I/O interface, they do not aim to protect application-level access pattern problems such as branch prediction attacks.

Application-Specific Data Oblivious Approaches. The application-specific approach requires developers to carefully examine all access patterns of a specific application and apply *oblivious operations* to protect them. Ohrimenko et al. [12] have analyzed a batch of well-known machine learning algorithms and identified that oblivious move (omove), oblivious greater(ogreater), and oblivious sorting (osort) are the three most frequently needed operations by these algorithms. omove and ogreater use the CMOV instructions to achieve obliviousness, which have been mentioned earlier. For experienced developers, this approach can thoroughly address all access-pattern problems. However, again, it is time-consuming and probably not practical for most HPC developers.

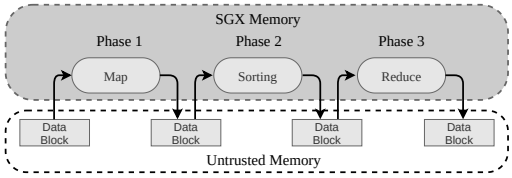

Figure 4. Regulated data flow between enclave and main memory (from SGX-MR [13])

Framework-Level Access Pattern Protection. A more promising solution is the *framework-level* protection scheme, such as SGX-MR [13], which achieve a balance between usability and access-pattern protection. The idea of SGX-MR is to regulate the application data flow with a framework, and then identify and protect the access patterns of the regulated data flow and the within-block (or page) access patterns. Once these access pattern problems are addressed, all applications using this framework will benefit. MapReduce is the right candidate for this purpose. Figure 4 shows how the application data flow is regulated by the MapReduce processing pipeline at the block level. (1) The input to the Map phase is just sequential reads, not leaking any information. (2) The Sorting phase can use an efficient *oblivious sorting* algorithm. (3) In the Reduce phase, we can protect the output privacy, i.e., group sizes. Users only need to implement the `map`, `reduce`, and possibly `combine`, functions, which only need to handle in-page branching statements – these are typically much easier to handle.

SGX-MR has unique advantages in transparency, programmability, efficiency, and attack protection. (1) *Transparency* is achieved with a carefully designed middle layer between TEE and user applications. It hides all the details about TEE processing and access-pattern protection. (2) It provides reasonably good *programmability*. Instead of emulating OS APIs, this approach utilizes the MapReduce processing framework to unify the applications at the framework

level. Almost all existing data mining and machine learning algorithms can be implemented with one or multiple MapReduce programs, as shown by Mahout and numerous examples during the past ten years with the booming big data applications. (3) This design can achieve better *efficiency* in access pattern protection than ORAM-based approaches [10], while the difficulty of redesigning users' applications is much lower than the customized data-oblivious approaches [12]. (4) The framework also allows users to achieve different levels of protection against application-oriented and memory page-oriented access-pattern attacks to meet various demands on performance and usability.

The goal of SGX-MR is not to provide a framework for legacy applications, as rewriting the code to use SGX-MR is not easy for most applications. It's more appropriate for those data-intensive applications that can be easily modified or developed from scratch. In addition, SGX-MR is still at the preliminary stage focusing on single-node processing. In contrast, multi-node processing is the norm for HPC applications. Opaque [19] has mentioned several access pattern issues with inter-nodes data exchange, which should be integrated into the extension of SGX-MR to multiple nodes.

Gaps: The above three approaches address the access pattern problem. (1) While new HPC applications can use these attack-mitigation methods, to our best knowledge, there is no solution to address side-channel attack problems for legacy HPC applications yet. In particular, as many HPC applications depend on scientific computing libraries, making these libraries fully data oblivious is very challenging. (2) ORAM and SGX-MR target data-intensive applications, but many HPC applications are compute-intensive, where new framework-level access-pattern protection methods should be developed. (3) All these data oblivious methods will impair performance, the level of which has not been fully understood yet for HPC applications.

Monitoring and Detecting Side-channel Attacks. This approach is attractive as it may avoid revising the existing codes (e.g., after using Graphene-SGX or SCONE to achieve good usability). The idea is to monitor the abnormal patterns of page fault interrupts or other system-level activities to detect possible attacks, as many attacks utilize these system features. While an attack is in progress, these system-level activities might differ from normal program execution. For example, SGX-TSX [20] has followed this approach. It utilizes Intel Transactional Synchronization Extensions (TSX) to monitor page-fault interrupts. TSX is a CPU built-in mechanism and cannot be compromised by attackers. Using TSX transactions, SGX-TSX detects anomalies and terminates the enclave programs as needed.

Gaps: The monitoring and detection approach is promising for protecting legacy code that cannot be easily modified. However, the current method: SGX-TSX is not easy to use and still requires a certain level of code modification. Another issue is false alarms, which might accidentally interrupt normal programs.

Blockchain-based Workflow Management. As we have discussed, collaborative HPC workflows may bring more challenges: dishonest owners, the conflict between confidentiality and provenance analysis, and the inconvenience in reproducibility verification. We envision a blockchain-based solution that can probably address most of these problems.

- *Protect from dishonest owners.* Use the blockchain to store the non-fungible signatures of the program and data. While this does not prohibit users from uploading fake data or algorithms or tampering with data and algorithms, we can trace the exact version used in a specific run.
- *Control accesses to provenance data.* Control the access to provenance analysis and log the accesses for anomaly detection. Access control can be reinforced with blockchain-maintained logs and smart contracts. We can also build an anomaly detection subsystem, learning from the tamper-resistant provenance access log.

The challenge is to develop an effective anomaly detection algorithm using the provenance access patterns. We can also prohibit access to the provenance data related to private components or their nearby components, which will significantly reduce the utility of provenance data.

- *Automated secure replay of workflows* can be implemented with smart contracts, which do not need owners of private components to stay online. It also prevents any attacks trying to compromise the integrity of reproducibility verification.

Gaps: (1) Blockchain applications are still in the embryonic stage. The cost of using current public blockchains is too high to be practical, while permissioned blockchains, such as HyperLedger, will need users to trust the management peers. (2) The access control and anomaly detection methods for provenance analysis will deter some attackers, but do not eliminate the risk of attack – they merely increase the chance of attackers getting caught.

Conclusion

HPC in the cloud can benefit many users who cannot own or access on-premise HPC resources. Recent studies have explored several aspects of HPC in the cloud, while the confidentiality issues have not been addressed yet. As data and computation confidentiality has been a general concern for many cloud users, we anticipate that HPC users may also have such needs in the future. Confidential computing has become practical due to the recent development of trusted execution environments, but it is still at the early stage of applications. We envision that combining TEE and HPC may raise some unique challenges, especially in a collaborative environment. We have analyzed the threat models for the single-user and collaborative-workflow cases, discussed several unique challenges, including usability, side-channel attacks, performance, and the interplay between confidential components and collaborative workflow, and reviewed some candidate solutions. We have also highlighted a few gaps that appear no satisfactory solutions yet, which probably indicate valuable research directions.

Acknowledgements

This work was supported in part by the U.S. National Science Foundation under Grant #2232824, Marquette University, and Northwestern Mutual Data Science Institute.

REFERENCES

1. M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha, and R. Buyya, "Hpc cloud for scientific and business applications: Taxonomy, vision, and research challenges," *ACM Comput. Surv.*, vol. 51, no. 1, 2018.
2. F. Khoda Parast, C. Sindhav, S. Nikam, H. Izadi Yekta, K. B. Kent, and S. Hakak, "Cloud computing security: A survey of service-based models," *Computers and Security*, vol. 114, 2022.
3. V. Costan and S. Devadas, "Intel sgx explained," *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.
4. S. Sagar and C. Keke, "Confidential machine learning on untrusted platforms: a survey," *Cybersecurity*, vol. 4, no. 1, p. 30, 2021. [Online]. Available: https://doi.org/10.1186/s42400-021-00092-8
5. A. Akram, A. Giannakou, V. Akella, J. Lowe-Power, and S. Peisert, "Performance analysis of scientific computing workloads on general purpose tees," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2021.
6. Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Innovations in Theoretical Computer Science Conference (ITSC)*, 2012, pp. 309–325.
7. Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *USENIX Conference on Security*, 2011, pp. 35–35.
8. S. Sharma and K. Chen, "Confidential boosting with random linear classifiers for outsourced user-generated data," in *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I*, 2019, pp. 41–65.
9. P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 19–38.
10. S. Sasy, S. Gorbunov, and C. W. Fletcher, "ZeroTrace : Oblivious memory primitives from intel SGX," in

Network and Distributed System Security Symposium, 2018.

11. Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, ser. SP '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 640–656. [Online]. Available: https://doi.org/10.1109/SP.2015.45
12. O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *USENIX Security Symposium*. USENIX Association, 2016, pp. 619–636.
13. A. M. Alam, S. Sharma, and K. Chen, "SGX-MR: Regulating dataflows for protecting access patterns of data-intensive sgx applications," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 1, pp. 5 – 20, 2021.
14. P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19.
15. M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *ACM Conference on Computer and Communications Security*, 2015.
16. F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *Proceedings of the 25th USENIX Conference on Security Symposium*, ser. SEC'16. USA: USENIX Association, 2016, pp. 601–618.
17. C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A practical library OS for unmodified applications on SGX," in *2017 USENIX Annual Technical Conference, USENIX ATC 2017, Santa Clara, CA, USA, July 12-14, 2017*, D. D. Silva and B. Ford, Eds., 2017, pp. 645–658.
18. S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure linux containers with intel sgx," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 689–703.
19. W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform," in *USENIX Symposium on Networked Systems Design and Implementation*, 2017.
20. M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-SGX: Eradicating controlled-channel attacks against enclave programs," in *Network and Distributed System Security Symposium 2017 (NDSS'17)*. Internet Society, February 2017.

Keke Chen is an associate professor with the Department of Computer Science at Marquette University and the Northwestern Mutual Data Science Institute. He directs the Trustworthy and Intelligent Computing Lab (TAIC). He earned his Ph.D. degree in Computer Science from Georgia Institute of Technology in 2006. His current research areas include confidential computing, data analytics, security&privacy of AI, and distributed computing. During 2006-2008, he was a senior research scientist at Yahoo! Labs, working on web search ranking, cross-domain ranking, and web-scale data mining. He owns three patents for his work at Yahoo! Labs. During 2008-2020, he was a faculty member with the Department of Computer Science and Engineering and the Center of Excellence in Knowledge-Enabled Computing at Wright State University. He is a senior IEEE member and an ACM member. His contact email is keke.chen@marquette.edu.